Shortest Paths III: All-pairs Shortest Paths, Dynamic Programming, Matrix Multiplication, Floyd-Warshall, Johnson

Lecture 16
Shortest paths

**Single-source shortest paths**
- Nonnegative edge weights
  - Dijkstra’s algorithm — $O(E + V \lg V)$
- General
  - Bellman-Ford — $O(VE)$
- DAG
  - One pass of Bellman-Ford $O(V + E)$

**All-pairs shortest paths**
- Nonnegative edge weights
  - Dijkstra’s algorithm $|V|$ times — $O(VE + V^2 \lg V)$
- General
  - Three algorithms today.
All-pairs shortest paths

**Input:** Digraph $G = (V, E)$, where $|V| = n$, with edge-weight function $w : V \rightarrow \mathbb{R}$.

**Output:** $n \times n$ matrix of shortest-path lengths $\delta(i, j)$ for all $i, j \in V$.

**Idea #1:**
- Run Bellman-Ford once from each vertex.
- Time $= O(V^2E)$.
- Dense graph $\Rightarrow O(V^4)$ time.

*Good first try!*
Dynamic programming

Consider the $n \times n$ adjacency matrix $A = (a_{ij})$ of the digraph, and define

$$d_{ij}^{(m)} = \text{weight of a shortest path from } i \text{ to } j \text{ that uses at most } m \text{ edges.}$$

**Claim:** We have

$$d_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j, \\ \infty & \text{if } i \neq j; \end{cases}$$

and for $m = 1, 2, \ldots, n - 1$,

$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}.$$
Proof of claim

\[ d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \} \]

Relaxation!

for \( k \leftarrow 1 \) to \( n \)

\[
\text{do if } d_{ij} > d_{ik} + a_{kj} \text{ then } d_{ij} \leftarrow d_{ik} + a_{kj}
\]

Note: No negative-weight cycles implies

\[ \delta(i, j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \ldots \]
Matrix multiplication

Compute \( C = A \cdot B \), where \( C, A, \) and \( B \) are \( n \times n \) matrices:

\[
c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}.
\]

Time = \( \Theta(n^3) \) using the standard algorithm.

What if we map “+” \( \rightarrow \) “\( \text{min} \)” and “\( \cdot \)” \( \rightarrow \) “+”?

\[
c_{ij} = \min_k \{a_{ik} + b_{kj}\}.
\]

Thus, \( D^{(m)} = D^{(m-1)} \times A \).

Identity matrix = \( I = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix} = D^0 = (d_{ij}^{(0)}) \).
Matrix multiplication (continued)

The \((\text{min, +})\) multiplication is \textit{associative}, and with the real numbers, it forms an algebraic structure called a \textit{closed semiring}.

Consequently, we can compute

\[
D^{(1)} = D^{(0)} \cdot A = A^1 \\
D^{(2)} = D^{(1)} \cdot A = A^2 \\
\vdots \\
D^{(n-1)} = D^{(n-2)} \cdot A = A^{n-1},
\]

yielding \(D^{(n-1)} = (\delta(i, j))\).

Time = \(\Theta(n \cdot n^3) = \Theta(n^4)\). No better than \(n \times \text{B-F}\).
Improved matrix multiplication algorithm

Repeated squaring: \( A^{2k} = A^k \times A^k \).
Compute \( A^2, A^4, \ldots, A^{2\left\lceil \log_2(n-1) \right\rceil} \).

\( \mathcal{O}(\log n) \) squarings

Note: \( A^{n-1} = A^n = A^{n+1} = \ldots \).

Time = \( \Theta(n^3 \log n) \).

To detect negative-weight cycles, check the diagonal for negative values in \( \mathcal{O}(n) \) additional time.
Floyd-Warshall algorithm

Also dynamic programming, but faster!

Define \( c_{ij}^{(k)} \) = weight of a shortest path from \( i \) to \( j \) with intermediate vertices belonging to the set \( \{1, 2, \ldots, k\} \).

Thus, \( \delta(i, j) = c_{ij}^{(n)} \). Also, \( c_{ij}^{(0)} = a_{ij} \).
Floyd-Warshall recurrence

\[ c_{ij}^{(k)} = \min_k \{ c_{ij}^{(k-1)}, c_{ik}^{(k-1)} + c_{kj}^{(k-1)} \} \]

intermediate vertices in \( \{1, 2, \ldots, k\} \)
Pseudocode for Floyd-Warshall

for \( k \leftarrow 1 \) to \( n \)
  do for \( i \leftarrow 1 \) to \( n \)
    do for \( j \leftarrow 1 \) to \( n \)
      do if \( c_{ij} > c_{ik} + c_{kj} \)
        then \( c_{ij} \leftarrow c_{ik} + c_{kj} \) \( \text{relaxation} \)

Notes:
• Okay to omit superscripts, since extra relaxations can’t hurt.
• Runs in \( \Omega(n^3) \) time.
• Simple to code.
• Efficient in practice.
Transitive closure of a directed graph

Compute \( t_{ij} = \begin{cases} 1 & \text{if there exists a path from } i \text{ to } j, \\ 0 & \text{otherwise.} \end{cases} \)

**IDEA:** Use Floyd-Warshall, but with \((\lor, \land)\) instead of \((\min, +)\):

\[
t_{ij}^{(k)} = t_{ij}^{(k-1)} \lor (t_{ik}^{(k-1)} \land t_{kj}^{(k-1)}).
\]

Time = \(\Theta(n^3)\).
Graph reweighting

**Theorem.** Given a label $h(v)$ for each $v \in V$, reweight each edge $(u, v) \in E$ by

$$\hat{w}(u, v) = w(u, v) + h(u) - h(v).$$

Then, all paths between the same two vertices are reweighted by the same amount.

**Proof.** Let $p = v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k$ be a path in the graph. Then, we have

$$\hat{w}(p) = \sum_{i=1}^{k-1} \hat{w}(v_i,v_{i+1})$$

$$= \sum_{i=1}^{k-1} \left( w(v_i,v_{i+1}) + h(v_i) - h(v_{i+1}) \right)$$

$$= \sum_{i=1}^{k-1} w(v_i,v_{i+1}) + h(v_k) - h(v_1)$$

$$= w(p) + h(v_k) - h(v_1).$$
Johnson’s algorithm

1. Find a vertex labeling $h$ such that $\hat{w}(u, v) \geq 0$ for all $(u, v) \in E$ by using Bellman-Ford to solve the difference constraints
   \[ h(v) - h(u) \leq w(u, v), \]
   or determine that a negative-weight cycle exists.
   • Time = $O(VE)$.

2. Run Dijkstra’s algorithm from each vertex using $\hat{w}$.
   • Time = $O(VE + V^2 \lg V)$.

3. Reweight each shortest-path length $\hat{w}(p)$ to produce the shortest-path lengths $w(p)$ of the original graph.
   • Time = $O(V^2)$.

Total time = $O(VE + V^2 \lg V)$. 